

Linear Hybrid System Falsification Through Descent^{*}

Houssam Abbas and Georgios Fainekos

Arizona State University, Tempe, AZ, USA,
`{hyabbas,fainekos}@asu.edu`

Abstract. In this paper, we address the problem of local search for the falsification of hybrid automata with affine dynamics. Namely, if we are given a sequence of locations and a maximum simulation time, we return the trajectory that comes the closest to the unsafe set. In order to solve this problem, we formulate it as a differentiable optimization problem which we solve using Sequential Quadratic Programming. The purpose of developing such a local search method is to combine it with high level stochastic optimization algorithms in order to falsify hybrid systems with complex discrete dynamics and high dimensional continuous spaces. Experimental results indicate that indeed the local search procedure improves upon the results of pure stochastic optimization algorithms.

Keywords: Model Validation and Analysis; Robustness; Simulation; Hybrid systems

1 Introduction

Despite the recent advances in the computation of reachable sets in medium to large-sized linear systems (about 500 continuous variables) [1, 2], the verification of hybrid systems through the computation of the reachable state space remains a challenging problem [3, 4]. To overcome this difficult problem, many researchers have looked into testing methodologies as an alternative. Testing methodologies can be coarsely divided into two categories: robust testing [5–7] and systematic/randomized testing [8–11].

Along the lines of randomized testing, we investigated the application of Monte Carlo techniques [12] and metaheuristics to the temporal logic falsification problem of hybrid systems. In detail, utilizing the robustness of temporal logic specifications [13] as a cost function, we managed to convert a decision problem, i.e., does there exist a trajectory that falsifies the system, into an optimization problem, i.e., what is the trajectory with the minimum robustness value? The resulting optimization problem is highly nonlinear and, in general, without any

^{*} This work was partially supported by a grant from the NSF Industry/University Cooperative Research Center (I/UCRC) on Embedded Systems at Arizona State University and NSF award CNS-1017074.

obvious structure. When faced with such difficult optimization problems, one way to provide an answer is to utilize some stochastic optimization algorithm like Simulated Annealing.

In our previous work [12], we treated the model of the hybrid system as a black box since a global property, such as convexity of the cost function, cannot be obtained, in general. One question that is immediately raised is whether we can use “local” information from the model of the system in order to provide some guidance to the stochastic optimization algorithm.

In this paper, we set the theoretical framework to provide local descent information to the stochastic optimization algorithm. Here, by local we mean the convergence to a local optimal point. In detail, we consider the falsification problem of affine dynamical systems and hybrid automata with affine dynamics where the uncertainty is in the initial conditions. In this case, the falsification problem reduces to an optimization problem where we are trying to find the trajectory that comes the closest to the unsafe set (in general, such a trajectory is not unique). A stochastic optimization algorithm for the falsification problem picks a point in the set of initial conditions, simulates the system for a bounded duration, computes the distance to the unsafe set and, then, decides on the next point in the set of initial conditions to try. Our goal in this paper is to provide assistance at exactly this last step. Namely, how do we pick the next point in the set of initial conditions? Note that we are essentially looking for a descent direction for the cost function in the set of initial conditions.

Our main contribution, in this paper, is an algorithm that can propose such descent directions. Given a test trajectory $s_{x_0} : \mathbb{R}_+ \mapsto \mathbb{R}^n$ starting from a point x_0 , the algorithm tries to find some vector d such that s_{x_0+d} gets closer to the unsafe set than s_{x_0} . We prove that it converges to a local minimum of the robustness function in the set of initial conditions, and demonstrate its advantages within a stochastic falsification algorithm. The results in this paper will enable local descent search for the satisfaction of arbitrary linear temporal logic specifications, not only safety specifications.

2 Problem Formulation

The results in this paper will focus on the model of hybrid automata with affine dynamics. A hybrid automaton is a mathematical model that captures systems that exhibit both discrete and continuous dynamics. In brief, a *hybrid automaton* is a tuple

$$\mathcal{H} = (X, L, E, Inv, Flow, Guard, Re)$$

where $X \subseteq \mathbb{R}^n$ is the state space of the system, L is the set of control locations, $E \subseteq L \times L$ is the set of control switches, $Inv : L \rightarrow 2^X$ assigns an invariant set to each location, $Flow : L \times X \rightarrow \mathbb{R}^n$ defines the time derivative of the continuous part of the state, $Guard : E \rightarrow 2^X$ is the guard condition that enables a control switch e and, finally, $Re : X \times E \rightarrow X \times L$ is a reset map. Finally, we let $H = L \times X$ to denote the state space of the hybrid automaton \mathcal{H} .

Formally, the semantics of a hybrid automaton are given in terms of generalized or timed transition systems [14]. For the purposes of this paper, we define a *trajectory* η_{h_0} starting from a point $h_0 \in H$ to be a function $\eta_{h_0} : \mathbb{R}_+ \rightarrow H$. In other words, the trajectory points to a pair of control location - continuous state vector for each point in time: $\eta_{h_0}(t) = (l(t), s_{x_0}(t))$, where $l(t)$ is the location at time t , and $s_{x_0}(t)$ is the continuous state at time t . We will denote by $\text{loc}(\eta_{h_0}) \in L^* \cup L^\omega$ the sequence of control locations that the trajectory η_{h_0} visits (no repetitions). The sequence is finite when we consider a compact time interval $[0, T]$ and η is not Zeno.

Assumptions: In the following, we make a number of assumptions. First, we assume that for each location $v \in L$ the system dynamics are affine, i.e., $\dot{x} = \text{Flow}(v, x) = Ax + b$, where A and b are matrices of appropriate dimensions. Second, we assume that the guards in a location are non-overlapping and that the transitions are taken as soon as possible. Thirdly, we assume that the hybrid automaton is deterministic, i.e., starting from some initial state, there exists a unique trajectory η_{h_0} of the automaton. This will permit us to use directly results from [6]. We also make the assumption that the simulation algorithms for hybrid systems are well behaved. That is, we assume that the numerical simulation returns a trajectory that remains close to the actual trajectory on a compact time interval. To avoid a digression into unnecessary technicalities, we will assume that both the set of initial conditions and the unsafe set are included in a single (potentially different) control location.

Let $\mathcal{U} \subseteq H$ be an unsafe set and let $D_{\mathcal{U}} : H \mapsto \mathbb{R}_+$ be the distance function to \mathcal{U} , defined by

$$D_{\mathcal{U}}(v, x) = \begin{cases} d_{\mathcal{U}}(x) & \text{if } v \in \text{pr}_L(\mathcal{U}) \\ +\infty & \text{otherwise} \end{cases}$$

where pr_L is the projection to the set of locations, pr_X is the projection to the continuous state-space and

$$d_{\mathcal{U}}(x) = \inf_{u \in \mathcal{U}} \|x - u\|.$$

Definition 1 (Robustness) *Given a compact time interval $[0, T]$, we define the robustness of a system trajectory η_h starting at some $h = (l, x) \in H$ to be $f(h) \triangleq \min_{0 \leq t \leq T} D_{\mathcal{U}}(\eta_h(t))$. When l is clear from the context, we'll write $f(x)$.*

Our goal in this paper is to find operating conditions for the system which produce trajectories of minimal robustness, as they indicate potentially unsafe operation. This can be seen as a 2-stage problem: first, decide on a sequence of locations to be followed by the trajectory. Second, out of all trajectories following this sequence of locations, find the trajectory of minimal robustness. This paper addresses the second stage. The central step is the solution the following problem:

Problem 1 *Given a hybrid automaton \mathcal{H} , a compact time interval $[0, T]$, a set of initial conditions $H_0 \subseteq H$ and a point $h_0 = (l_0, x_0) \in H_0$ such that $0 < f(h_0) < +\infty$, find a vector dx such that $h'_0 = (l_0, x_0 + dx)$, $\text{loc}(\eta_{h_0}) = \text{loc}(\eta_{h'_0})$ and $f(h'_0) \leq f(h_0)$.*

An efficient solution to Problem 1 may substantially increase the performance of the stochastic falsification algorithms by proposing search directions where the robustness decreases. In summary, our contributions are:

- We formulate Problem 1 as a nonlinear optimization problem, which we prove to be differentiable w.r.t. the initial conditions. Thus it is solvable with standard optimizers.
- We developed an algorithm, Algorithm 1, to find local minima of the robustness function.
- We demonstrate the use of Algorithm 1 in a higher-level stochastic falsification algorithm, and present experimental results to analyze its competitiveness against existing methods.

3 Finding a descent direction

Consider an affine dynamical system in \mathbb{R}^n ,

$$\dot{x} = F(x) = Ax + b$$

which we assume has a unique solution

$$s_{x_0}(t) = e^{At}x_0 + c(t)$$

where $x_0 \in X_0$ is the initial state of the trajectory

Let $\mathcal{U} \subset \mathbb{R}^n$ be the *convex* set of bad states, and $\overline{\mathcal{U}}$ its closure. Note that even for linear systems, $f : X_0 \mapsto \mathbb{R}_+$ is not necessarily differentiable or convex. Our goal is to find the trajectory of minimum robustness. That is done by a local search over the set of initial conditions.

Given an initial state x_0 and a trajectory s_{x_0} that starts at x_0 , define the time t^* of its closest proximity to \mathcal{U} , and the point $u^* \in \overline{\mathcal{U}}$ which is closest to the trajectory:

$$t^* = \arg \min_{t \geq 0} d_{\mathcal{U}}(s_{x_0}(t)), u^* = \arg \min_{u \in \overline{\mathcal{U}}} \|s_{x_0}(t^*) - u\|$$

3.1 Partial descent based at the nearest point

Given t^* , choose an *approach vector* d' such that $s_{x_0}(t^*) + d'$ is closer to \mathcal{U} than $s_{x_0}(t^*)$. Such a vector always exists given that s_{x_0} has a positive distance to \mathcal{U} . Moreover, it is not unique. Thus we have

$$f(x_0) = \|s_{x_0}(t^*) - u^*\| > \min_u \|s_{x_0}(t^*) + d' - u\|$$

Define $d = e^{-At^*} d'$. Then

$$\begin{aligned} f(x_0) &> \min_u \|s_{x_0}(t^*) + d' - u\| = \min_u \|e^{At^*} x_0 + c(t^*) + e^{At^*} d - u\| \\ &\geq \min_t \min_u \|(x_0 + d)e^{At} + c(t) - u\| = f(x_0 + d) \geq 0 \end{aligned}$$

and d is a descent direction, provided that $x_0 + d \in X_0$.

It is easy to see that for any $x_0 \in X_0$ and $d' \in \mathbb{R}^n$,

$$s_{x_0 + e^{-At^*} d'}(t) = s_{x_0}(t^*) + d'$$

so the new distance is achieved at the same time t^* as the old one. This new distance $d_{\mathcal{U}}(s_{x_0+d}(t^*))$ is an upper *bound* on the new trajectory's robustness. In general, the new trajectory's robustness might be even smaller, and achieved at some other time $t' \neq t^*$.

As pointed out earlier, the approach d' is not unique. The requirement on d' is that $s_{x_0}(t^*) + d'$ be closer to \mathcal{U} than $s_{x_0}(t^*)$. So define the set $P(x_0; t^*)$ of points that are closer to \mathcal{U} than $s_{x_0}(t^*)$ (see Fig. 1):

$$P(x_0; t^*) \triangleq \{x \in \mathbb{R}^n \mid d_{\mathcal{U}}(x) \leq f(x_0)\} \quad (1)$$

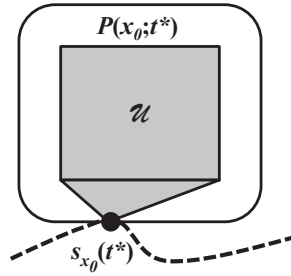


Fig. 1. The unsafe set \mathcal{U} and the set $P(x_0; t^*)$. The system trajectory s_{x_0} appears as a dashed curve.

Then d' must satisfy $s_{x_0}(t^*) + d' \in P(x_0; t^*) \Leftrightarrow d \in e^{-At^*}(P(x_0; t^*) - s_{x_0}(t^*))$. Combined with the requirement that $x_0 + d \in X_0$, we get

$$d \in (X_0 - x_0) \cap e^{-At^*}[P(x_0; t^*) - s_{x_0}(t^*)]$$

Any point in the above *descent set* is a feasible descent direction. As a special case, it is easy to verify that $d = e^{-At^*}(\bar{u} - s_{x_0}(t^*))$, for any $\bar{u} \in \bar{\mathcal{U}}$, is a descent direction that leads to 0 robustness. Coupled with the requirement that $x_0 + d$ must be in X_0 , it comes

$$d \in (X_0 - x_0) \cap e^{-At^*}(\bar{\mathcal{U}} - s_{x_0}(t^*))$$

If computing P is too hard, we can approximate it with the following \mathcal{U}^U : imagine translating \mathcal{U} along the direction $v = s_{x_0}(t^*) - u^*$, so it is being drawn closer to $s_{x_0}(t^*)$, until it meets it. Then we claim that the union of all these translates forms a set of points closer to \mathcal{U} than $s_{x_0}(t^*)$:

Proposition 1. *Let \mathcal{U} be a convex set, $s_{x_0}(t^*)$ a point outside it, and $\mathcal{U}^U(v)$ be the Minkowski sum of \mathcal{U} and $\{\alpha v | \alpha \in [0, 1]\}$. Then for any p in $\mathcal{U}^U(v)$, $d_{\mathcal{U}}(p) \leq d_{\mathcal{U}}(s_{x_0}(t^*))$*

Proof. \mathcal{U}^U is convex by the properties of Minkowski sums. Let $\bar{u} \in \partial\mathcal{U}$. Then for any $\alpha \leq 1$, $d_{\mathcal{U}}(\bar{u} + \alpha v) \leq \|\bar{u} + \alpha v - \bar{u}\| = \alpha\|v\| = \alpha f(x_0) \leq f(x_0)$. So translates of boundary points are closer to \mathcal{U} than $s_{x_0}(t^*)$.

Now we show that all points in $\mathcal{U}^U/\mathcal{U}$ are translates of boundary points. Consider any point $p = u + \alpha v$ in $\mathcal{U}^U/\mathcal{U}$: u is in \mathcal{U} , but p is not, so the line $[u, p]$ crosses $\partial\mathcal{U}$ for some value α^o : $u + \alpha^o v \in \partial\mathcal{U}$. And, $p = u + \alpha v = (u + \alpha^o v) + (\alpha - \alpha^o)v$, so by what preceded, $d_{\mathcal{U}}(p) \leq f(x)$.

When $p \in \mathcal{U}$, of course, $d_{\mathcal{U}}(p) = 0 \leq f(x)$.

We have thus defined 3 possible descent sets: $\mathcal{U} \subset \mathcal{U}^U \subset P(x_0; t^*)$.

3.2 Implementation

The question we address here is: how do we obtain, *computationally*, points in the descent set \mathcal{W} , where $\mathcal{W} = \mathcal{U}, P(x_0)$ or $\mathcal{U}^U(v)$? The following discussion is based on Chapters 8 and 11 of [15].

Since we're assuming X_0 and \mathcal{U} to be convex, then the descent set is also convex. Describe X_0 with a set of N_X inequalities $q_i(x) \leq 0$ where the q_i are convex and differentiable, and $\mathcal{W} = \{x | p_i(x; x_0) \leq 0, i = 1 \dots k\}$ for convex differentiable p_i (the particular form of the p_i will depend on the descent set at hand). We assume $\text{dom } p_i = \text{dom } q_i \triangleq \mathbb{R}^n$.

Given an already simulated trajectory s_{x_0} and its time of minimum robustness t^* , we are looking for a feasible x_1 such that $s_{x_1}(t) \in \mathcal{W}$ for some t . Thus we want to solve the following feasibility problem

$$\begin{aligned} \min_{(x, \nu)} \quad & \nu \\ \text{s.t.} \quad & p_i(s_x(t); x_0) \leq \nu, i = 1 \dots k \quad (\text{t-PDP}(x_0)) \\ & q_i(x) \leq \nu, i = 1 \dots N_X \end{aligned} \tag{2}$$

This is a convex program, which can be solved by a Phase I Interior Point method [15]. A non-positive minimum ν^* means we found a feasible x ; if $\mathcal{W} = \mathcal{U}$, then our work is done: we have found an unsafe point. Else, we can't just stop upon finding a non-positive minimum: we have merely found a new point x_1 whose robustness is less than x_0 's, but not (necessarily) 0. So we iterate: solve t-PDP(x_0) to get x_1 , solve t-PDP(x_1) to get x_2 , and so on, until $f(x_i) = 0$, a maximum number of iterations is reached, or the problem is unsolvable. If the minimum is positive, this means that *for this value of t* , it is not possible for any trajectory to enter $\bar{\mathcal{U}}$ at time t .

The program suffers from an arbitrary choice of t . One approach is to sample the trajectory at a fixed number of times, and solve (2) for each. This is used in the experiments of this section. A second approach, used in the next section, is to let the optimization itself choose the time, by adding it to the optimization variable. The resulting program is no longer necessarily convex.

3.3 Numerical Experiments

In this section, we present some numerical experiments demonstrating the practical significance of the previous theoretical results.

Example 1 We consider the verification problem of a transmission line [16]. The goal is to check that the transient behavior of a long transmission line has acceptable overshoot for a wide range of initial conditions. Figure 2 shows a model of the transmission line, which consists of a number of RLC components (R: resistor, L: inductor and C: capacitor) modeling segments of the line. The left side is the sending end and the right side is the receiving end of the transmission line.

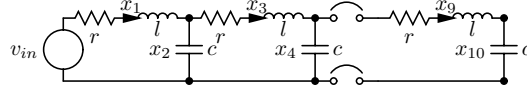


Fig. 2. RLC model of a transmission line.

The dynamics of the system are given by a linear dynamical system

$$\dot{x}(t) = Ax(t) + bV_{in}(t) \text{ and } V_{out}(t) = Cx(t)$$

where $x(t) \in \mathbb{R}^{81}$ is the state vector containing the voltage of the capacitors and the current of the inductors and $V_{in}(t) \in \mathbb{R}$ is the voltage at the sending end. The output of the system is the voltage $V_{out}(t) \in \mathbb{R}$ at the receiving end. Here, A , b and C are matrices of appropriate dimensions. Initially, we assume that the system might be in any operating condition such that $x(0) \in [-0.1, 0.1]^{41} \times [-0.01, 0.01]^{40}$. Then, at time $t = 0$ the input is set to the value $V_{in}(t) = 1$.

The descent algorithm is applied to the test trajectory that starts from $x(0) = 0$ and it successfully returns a trajectory that falsifies the system (see Fig. 3).

4 Hybrid systems with affine dynamics

We now turn to the case of hybrid systems with affine dynamics in each location. The objective is still to find a descent direction in H_0 , given a simulated trajectory η_{h_0} originating at point $h_0 \in H_0$. Note that since we have assumed that $\text{pr}_L(H_0)$ is a singleton set, the problem reduces to finding a descent direction in $X_0 = \text{pr}_X(H_0)$.

Assumptions. At this point, we make the following assumptions:

- a. The continuous dynamics in each location are stable.¹

¹ This is not a restrictive assumption since we can also consider incrementally stable systems [17], and even unstable linear systems [18].

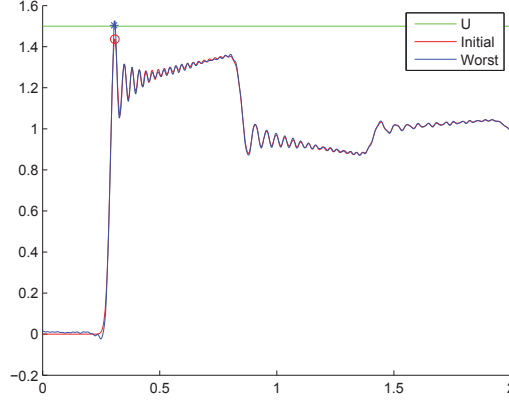


Fig. 3. The unsafe set U , the initial test trajectory starting from $x(0) = 0$ and the trajectory that falsifies the system.

b. For every transition $e \in L^2$, the resets $Re(\cdot, e)$ are differentiable functions of their first argument.

c. Conditions 4 and 5 of Theorem III.2 in [19] are satisfied, namely: for all i , there exists a differentiable function $\sigma_i : \mathbb{R}^n \mapsto \mathbb{R}$ such that $Inv(l_i) = \{x \in \mathbb{R}^n | \sigma_i(x) \geq 0\}$; and, for all i, x such that $\sigma_i(x) = 0$, the Lie derivative $L_F \sigma_i(x) \neq 0$. This allows us to have a *differentiable* transition time t_x of the trajectory starting at the initial point $x \in X_0$.

d. The sequence of locations $loc(\eta_{h_0})$ enters the location of the unsafe set. This is required for our problem to be well-defined (specifically, for the objective function to have finite values). The task of finding such an h_0 is delegated to the higher-level stochastic search algorithm, within which our method is integrated.

4.1 Descent in the Robustness Ellipsoid

Consider a trajectory η_{h_0} with positive robustness, with $loc(\eta_{h_0}) = l_0 l_1 \dots l_N$. This is provided by the simulation. Let the initial set X_0 be in location l_0 and let $l_{\mathcal{U}}$ denote the location of \mathcal{U} . In order to solve Problem 1, we assume that $l_{\mathcal{U}}$ appears in $loc(\eta_{h_0})$ (see Assumption d above) - otherwise, $f(h_0) = +\infty$ and the problem as posed here is ill-defined. We search for an initial point $h'_0 \in H_0$ (actually $x'_0 \in X_0$), whose trajectory gets closer to the unsafe set than the current trajectory η_{h_0} .

In order to satisfy the constraints of Problem 1, we need to make sure that the new point h'_0 that we propose generates a trajectory that follows the same sequence of locations as η_{h_0} . This constraint can be satisfied using the notion of robust neighborhoods introduced in [6]. In [6], it is shown that for stable systems and for a given safe initial point $h_0 = (l_0, x_0)$, there exists an ‘ellipsoid of robustness’ centered on x_0 , such that any trajectory starting in the ellipsoid,

remains in a tube around η_{h_0} . The tube has the property that all trajectories in it follow the same sequence of locations as η_{h_0} . Therefore, we restrict the choice of initial point to $X_0 \cap E(x_0)$, where $E(y) = \{x | (x - y)^T R^{-1} (x - y) \leq 1\}$ is the ellipsoid of robustness centered on x_0 , with shape matrix R . Formally, in [6], the following result was proven.

Theorem 1 *Consider a hybrid automaton \mathcal{H} , a compact time interval $[0, T]$, a set of initial conditions $H_0 \subseteq H$ and a point $h_0 = (l_0, x_0) \in H_0$. Then, we can compute a number $\varepsilon > 0$ and a bisimulation function $\phi(x_1, x_2) = (x_1 - x_2)^T M (x_1 - x_2)$, where M is a positive semidefinite matrix, such that for any $x'_0 \in \{y \in X \mid \phi(x_0, y) \leq \varepsilon\}$, we have $\text{loc}(\eta_{h_0}) = \text{loc}(\eta_{(l_0, x'_0)})$.*

Remark 1 (i) In [6], in the computation of ε , we also make sure that any point in the robust neighborhood generates a trajectory that does not enter the unsafe set. In this work, we relax this restriction since our goal is to find a point that generates a trajectory that might enter the unsafe set. (ii) In view of Theorem 1, the shape matrix for the ellipsoid is defined as $R = \varepsilon^2 M^{-1}$.

We now proceed to pose our search problem as a feasibility problem. Let t_0 be the time at which s_{x_0} is closest to \mathcal{U} . We choose $P(x_0; t_0)$ as our descent set: recall that it is the set of all points which are closer to \mathcal{U} than $s_{x_0}(t_0)$ (Def. 1). Therefore, if we can find $x^* \in X_0 \cap E(x_0)$ such that $s_{x^*}(t^*) \in P(x_0; t_0)$ for some t^* , it follows that $f(x^*) \leq f(x_0)$. To simplify notation, let $\mathcal{W} = P(x_0; t_0)$ be the descent set. As before, it is assumed that $\mathcal{W} = \{x \in \mathbb{R}^n : p_i(x) \leq 0, i = 1 \dots k\}$ for differentiable p_i . The search problem becomes:

Given η_{h_0} , find $x^* \in X_0 \cap E(x_0)$ and $t^* \geq 0$, such that $s_{x^*}(t^*) \in \mathcal{W}$. This is cast as an optimization problem over $z \in \mathbb{R}^n \times \mathbb{R}_+ \times \mathbb{R}$:

$$\begin{aligned} \min_{z=(x,t,\nu)} \quad & \nu \\ \text{s.t.} \quad & C_0 x - g_0 \leq 0 \\ & (x - x_0)^T P^{-1} (x - x_0) - 1 \leq \nu \\ & p_i(s_x(t); x_0) \leq \nu, i = 1 \dots k \end{aligned} \tag{3}$$

where $s_x(t) = \text{pr}_X(\eta_{(l_0, x)}(t))$ and $X_0 = \{x | C_0 x - g_0 \leq 0\}$.

Remark 2 *Note that Problem (3) is specific to a choice of initial point x_0 ; this will be important in what follows. In our implementation, the first constraint is specified as bounds to the optimization and so is always satisfied.*

Later in this section, we discuss how to solve this optimization problem. For now, we show how solving this problem produces a descent direction for the robustness function. For convenience, for $z = (x, t, \nu)$, we define the constraint

functions

$$G_0(z) = C_0 x - g_0 \quad (4a)$$

$$G_E(z) = (x - x_0)^T P^{-1} (x - x_0) - 1 \quad (4b)$$

$$G_{\mathcal{W}}(z) = \begin{pmatrix} p_1(s_x(t); x_0) \\ \vdots \\ p_k(s_x(t); x_0) \end{pmatrix} \quad (4c)$$

A point z is *feasible* if it satisfies the constraints in Problem (3). Finally, define the objective function $F(z) = \nu$.

The objective function $F(z)$ measures the *slack* in satisfying the constraints: a negative ν means all constraints are strictly satisfied, and in particular, $G_{\mathcal{W}}$. Thus, we have a trajectory that enters \mathcal{W} and, hence, gets strictly closer to \mathcal{U} . This reasoning is formalized in the following proposition:

Proposition 2. *Let $z^* = (x^*, t^*, \nu^*)$ be a minimum of $F(z)$ in program (3). Then $f(l_0, x^*) \leq f(l_0, x_0)$.*

Proof. It is assumed that the optimizer is iterative and that it returns a solution that decreases the objective function. In what follows, for a vector $y \in \mathbb{R}^n$, $\max y$ is the largest entry in y .

We first remark that for a given x and t that satisfy the constraints in (3),

$$z = (x, t, \max\{G_E(x, t), G_{\mathcal{W}}(x, t)\})$$

is feasible, and $F(z) \leq F(x, t, \nu)$ for any feasible (x, t, ν) . Therefore, we may only consider points with $F(z) = \nu = \max\{G_E(x, t), G_{\mathcal{W}}(x, t)\}$.

Let $z_0 = (x_0, t_0, \nu_0)$ be the initial point of the optimization. Because x_0 is the center of $E(x_0)$, $G_E(z_0) = -1$. And, because $s_{x_0}(t_0) \in \partial\mathcal{W}$, $\max G_{\mathcal{W}}(z_0) = 0$. Thus $\nu_0 = 0$. Therefore, at the minimum $z^* = (x^*, t^*, \nu^*)$ returned by the optimizer, $\nu^* \leq \nu_0 = 0$. In particular, $G_{\mathcal{W}}(z^*) \leq 0$, and the new trajectory s_{x^*} enters \mathcal{W} . Therefore, its robustness is no larger than that of the initial trajectory s_{x_0} .

We now address how Problem 3 might be solved. Functions F , G_0 and G_E are differentiable in $z = (x, t, \nu)$. It is not clear that $G_{\mathcal{W}}$, or equivalently, $p_i(s_x(t); x_0)$, as a function of z , is differentiable. We now show that under some assumptions on the p_i , for trajectories of linear systems, p_i is in fact differentiable in both x and t , over an appropriate range of t . This implies differentiability in z . Therefore, standard gradient-based optimizers can be used to solve Problem 3.

For the remainder of this section, we will re-write $s_x(t)$ as $s(x, t)$ to emphasize the dependence on the initial point x . $s^{(i)}(x, \tau)$ will denote the point, at time τ , on the trajectory starting at $x \in \text{Inv}(l_i)$, and evolving according to the dynamics of location i . When appearing inside location-specific trajectories such as $s^{(i)}$, the time variable will be denoted by the greek letter τ to indicate *relative* time: that

is, time measured from the moment the trajectory entered l_i , not from datum 0. $s(x, t)$ (without superscript) will denote the hybrid trajectory, traversing one or more locations. We will also drop the x_0 from $p_i(y; x_0)$, and write it simply as $p_i(y)$.

We first prove differentiability in x . Therefore, unless explicitly stated otherwise, the term ‘differentiable’ will mean ‘differentiable in x ’. Start by noting that $p_i(s(x, t))$ is a composite function of $x \in X_0$. Since p_i is differentiable, it is sufficient to prove that $s(x, t)$ is differentiable. The hybrid trajectory $s(x, \cdot)$ is itself the result of composing the dynamics from the visited locations l_0, \dots, l_{N-1} .

Recall that $E(x_0)$ is the ellipsoid of robustness centered at x_0 . As shown by Julius et al. [6], the following times are well-defined:

Definition 2 (Transition times) *Given $x_0 \in X_0$, let $E_0 \triangleq \text{int}(E(x_0) \cap X_0)$. t_i is the time at which trajectory $s(x_0)$ transitions from $\text{Inv}(l_{i-1})$ into $\text{Inv}(l_i)$ through guard $\text{Guard}(l_{i-1}, l_i)$. t_i^- is the maximal time for which the image of E_0 under the hybrid dynamics is contained in $\text{Inv}(l_{i-1})$:*

$$t_i^- = \max\{t | s(E_0, t) \subset \text{Inv}(l_{i-1})\}$$

In other words, t_i^- is the time at which occurs the first l_{i-1} -to- l_i transition of a point in $s(E_0)$.

t_i^+ is the minimal time for which the image of E_0 under the hybrid dynamics is contained in $\text{Inv}(l_i)$:

$$t_i^+ = \min\{t | s(E_0, t) \subset \text{Inv}(l_i)\}$$

In other words, t_i^+ is the time at which occurs the last l_{i-1} -to- l_i transition of a point in $s(E_0)$.

For a given point $x \in X_0$, $t_x^{i-1 \rightarrow i}$ ($\tau_x^{i-1 \rightarrow i}$) is the absolute (relative) transition time of trajectory $s^{(i)}(x)$ from $\text{Inv}(l_{i-1})$ into $\text{Inv}(l_i)$ through guard $\text{Guard}(l_{i-1}, l_i)$. Thus, for example, $t_1 = t_{x_0}^{0 \rightarrow 1} = \tau_{x_0}^{0 \rightarrow 1}$ and $t_2 = t_{x_0}^{1 \rightarrow 2} = \tau_{x_0}^{0 \rightarrow 1} + \tau_{y_0}^{1 \rightarrow 2}$, with $y_0 = s^0(x_0, t_{x_0}^{0 \rightarrow 1})$. When the transition is clear from context, we will simply write t_x (τ_x).

We will first show differentiability of a trajectory that visits only 2 locations l_0 and l_1 :

$$s(x_0, t) = s^{(1)}(Re(s^{(0)}(x_0, t_x), (l_0, l_1)), t - t_x) \quad (5)$$

Example 2 *We first present a simple 1D example to illustrate the definitions and the idea of the proof. Consider the hybrid system with three locations*

$$\mathcal{H} = (\mathbb{R}, \{0, 1, 2\}, \{(0, 1), (1, 2)\}, \text{Inv}, \text{Flow}, \text{Guard}, \text{Id})$$

where $\text{Inv}(l) = \mathbb{R}$ for $l = 0, 1, 2$, and the flow is defined by

$$\text{Flow}(l, x) = \dot{x}(t) \begin{cases} x(t) & \text{if } l \in \{0, 2\} \\ -x(t) & \text{if } l = 1 \end{cases}$$

The guards are $\text{Guard}(0,1) = \{1\}$ and $\text{Guard}(1,2) = \{1/4\}$. Id is the identity map, so there are no resets. The initial set is $X_0 = [0, 1/2]$. The solutions in the individual locations are then

$$s^{(0)}(x, t) = e^t x$$

$$s^{(1)}(x, t) = e^{-t} x$$

$$s^{(2)}(x, t) = e^t x$$

We can solve, in this simple case, for $\tau_x^{0 \rightarrow 1}: e^{\tau_x} x = 1 \Rightarrow \tau_x^{0 \rightarrow 1} = \ln(1/x)$. Similarly for $\tau_x^{1 \rightarrow 2}: e^{-\tau_x} \cdot 1 = 1/4 \Rightarrow \tau_x^{1 \rightarrow 2} = \ln(4x)$.

We first show differentiability of the trajectory over locations 0 and 1. We then do the same for a trajectory over locations 1 and 2. Then we stitch the two together and show differentiability over 3 locations. For locations 0 and 1: $s(x, t) = s^{(1)}(s^{(0)}(x, t_x), t - t_x) = s^{(1)}(1, t - t_x) = e^{-(t-t_x)} \cdot 1 = e^{-t}/x \Rightarrow \frac{d}{dt} s(x, t) = -\frac{e^{-t}}{x^2}$.

Moving on the trajectory over locations 1 and 2, the procedure is the same: from an initial point $x \in \text{Guard}(0,1) = \{1\}$, for a fixed (relative time) $\tau \in (t_2 - t_1, t_3^- - t_1): s(x, \tau) = s^{(2)}(s^{(1)}(x, \tau_x), \tau - \tau_x) = s^{(2)}(1/4, \tau + \ln(1/4x)) = e^{\tau + \ln(1/4x)} 1/4 = e^\tau / 16x \Rightarrow \frac{d}{d\tau} s(x, \tau) = -\frac{e^\tau}{16x^2}$.

Finally we stitch up the 2 portions of the trajectory: $x \in X_0, t \in [t_2, t_3^-]$. $s(x, t) = s^{(2)}(s^{(1)}(s^{(0)}(x, t_1), t_2 - t_1), t - t_2) = s^{(2)}(s^{(1)}(1, t_2 - t_1), t - t_2) = s^{(2)}(1/4, t - t_2) = e^{t-t_2}/4$. Since $t_2 = t_x^{0 \rightarrow 1} + \tau_1^{1 \rightarrow 2} = \ln(1/x) + \ln(4 \cdot 1) = \ln(4/x) \Rightarrow s(x, t) = \frac{e^t}{4} e^{\ln(x/4)} = x e^t / 16 \Rightarrow \frac{d}{dt} s(x, t) = e^t / 16$.

We now prove the general case.

Proposition 3. Let $x_0 \in E_0$, and fix $t \in (t_1, t_2^-]$. Consider the hybrid trajectory over 2 locations in Eq.(5). If Assumptions a-d are satisfied, then $s(x, t)$ is differentiable at x_0 .

Proof. In what follows, $e = (l_0, l_1)$.

$$\begin{aligned} s^{(0)}(x, \tau_x) &= e^{\tau_x A_0} x + \int_0^{\tau_x} e^{(\tau_x - s) A_0} b ds \\ &= \underbrace{e^{\tau_x A_0} x}_{\text{term1}} + \underbrace{e^{\tau_x A_0} \int_0^{\tau_x} e^{-s A_0} b ds}_{\text{term3}} \end{aligned}$$

Terms 1 and 2 are clearly differentiable in x . For term3, write $M(t) = \int_0^t e^{-s A_0} b ds$ so term3 = $M(\tau_x)$. $M(t)$ is differentiable by the 2nd Fundamental Theorem of Calculus and its derivative is $M'(t) = e^{-t A_0} b$. As a consequence of Assumption c, τ_x itself is differentiable in x (Lemma III.3 in [19]), and the chain rule allows us to conclude that term3 is differentiable in x . Thus $s^{(0)}(x, \tau_x)$ is differentiable over E_0 . Since $\text{Re}(\cdot, e)$ is differentiable by Assumption b, then

$Re(s^{(0)}(x, \tau_x), e)$ is differentiable over E_0 . Note that E_0 is open and $s^{(0)}$ is continuous, so $U = \{w \in \mathbb{R}^n | w = s^{(0)}(x, t_x) \text{ for some } x \in E_0\} \subset Guard(e)$ is open. Since $Re(\cdot, e)$ is continuous, then $Re(U, e)$ is open. Next,

$$\begin{aligned} s(x, t) &= s^{(1)}(Re(s^{(0)}(x, t_x), e), t - t_x) \\ &= \underbrace{e^{(t-t_x)A_1}}_{term4} Re(s^{(0)}(x, t_x), e) + \underbrace{e^{(t-t_x)A_1}}_{term5} \underbrace{\int_0^{t-t_x} e^{-sA_1} b_1 ds}_{term6} \end{aligned}$$

Using the same argument as above, terms 4, 5 and 6 are differentiable in x . In conclusion, $s(x, t)$ is differentiable at over E_0 , and this ends the proof.

The following proposition generalizes Prop. 3 to trajectories over more than 2 locations.

Proposition 4. *Fix $t \in (t_{N-1}, T]$, and consider the hybrid trajectory over $N \geq 1$ locations. Then $s(x, t)$ is differentiable at x_0 for all $x_0 \in E_0$.*

Proof. We argue by induction over the number of locations N . The base case $N = 1$ is true by hypothesis, and the case $N = 2$ has been proven in Prop. 3. For $N > 2$ and $t \leq t_{N-1}$, let $\zeta(x, t)$ be the trajectory over the first $N - 1$ locations, so that $s(x, t) = s^{(N-1)}(Re(\zeta(x, \tau_{N-2}), (l_{N-2}, l_{N-1})), t - t_{N-1})$. By the induction hypothesis, $\zeta(x, t)$ is differentiable at x_0 . Then ζ and $s^{(N-1)}$ satisfy the conditions of the case $N = 2$.

Differentiability with respect to time is easily proven:

Proposition 5. *Let $x_0 \in E_0$ and $t \in (t_{N-1}, T)$, that is, a time at which the trajectory is in the last location. Consider the hybrid trajectory over $N \geq 1$ locations. Then $s(x_0, t)$ is differentiable in t over $[t_{N-1}, T)$.*

Proof. $s(x_0, t) = s^{(N-1)}(x_0, t - t_{N-1})$. The location-specific trajectories $s^{(i)}(x, \cdot)$ are solutions of differential equations involving at least the first time derivative. Therefore, they are smooth over (t_{N-1}, T) . This implies differentiability of the hybrid trajectory $s(x_0, \cdot)$ over the same interval. At $t = T$, the trajectory is only left-differentiable, since it's undefined from the right.

The following result is now a trivial application of the chain rule to $p_i \circ s$:

Proposition 6. *Let $x_0 \in E_0$, $t \in (t_{N-1}, T)$. If p_i is differentiable for all $i = 1, \dots, k$, then $G_{\mathcal{W}}$ is differentiable in z over $E_0 \times \mathbb{R}_+ \times \mathbb{R}$.*

We choose Sequential Quadratic Programming (SQP), as a good general-purpose optimizer to solve Problem 3. SQP is a Q-quadratically convergent iterative algorithm. At each iterate, $G_{\mathcal{W}}(x_i, t_i, \nu_i)$ is computed by simulating the system at x_i . This is the main computational bottleneck of this method, and will be discussed in more detail in the Experiments section.

Algorithm 1 Robustness Ellipsoid Descent (RED)

Input: An initial point $x_0 \in X_0$, and corresponding t_0 .

Output: z_Q .

```
1: Initialization:  $i = 0$ 
2: Compute  $z_i^* = (x_i^*, t_i^*, \nu_i^*) = \text{minimum of Prob3}[\mathcal{W}_i]$ .
3: while  $\nu_i^* < 0$  do
4:    $x_{i+1} \leftarrow x_i^*$ 
5:    $t_{i+1} = \arg \min_t d_{\mathcal{U}}(s_{x_{i+1}}(t))$ 
6:    $\mathcal{W}_{i+1} = P(x_{i+1})$ 
7:   Compute  $z_i^* = (x_i^*, t_i^*, \nu_i^*) = \min \text{ of Prob3}[\mathcal{W}_{i+1}]$ .
8:    $i = i + 1$ 
9: end while
10:
11: Return  $z_Q \triangleq z_i^*$ 
```

4.2 Convergence to a local minimum

Solving Problem (3), for a given \mathcal{W} , produces a descent direction for the robustness function. However, one can produce examples where a local minimum of $F(\cdot)$ is not a local minimum of the robustness function f . This section derives conditions under which *repeated* solution of Problem (3) yields a local minimum of the robustness function.

For $i = 0, 1, 2, \dots$, let $x_i \in X_0 \cap E(x_{i-1})$, and let t_i be the time when s_{x_i} is closest to \mathcal{U} . Let $\mathcal{W}_i = P(x_i; t_i)$ be the descent set for this trajectory. For each \mathcal{W}_i , one can setup the optimization Problem (3) with $\mathcal{W} = \mathcal{W}_i$, and initial point $(x_i, t_i, 0)$; this problem is denoted by $\text{Prob3}[\mathcal{W}_i]$. (Recall from the proof of Proposition 2 that $\nu = 0$ at the initial point of the optimization problem). Finally, let $z_i^* = (x_i^*, t_i^*, \nu_i^*)$ be the minimum obtained by solving $\text{Prob3}[\mathcal{W}_i]$.

Algorithm 1 describes how to setup a sequence of optimization problems that leads to a local minimum of f . It is called Robustness Ellipsoid Descent, or RED for short.

Proposition 7. *Algorithm 1 (RED) terminates*

Proof. Proposition 2 holds for each problem $\text{Prob3}[\mathcal{W}_i]$. Therefore, each solution with $\nu_i < 0$ gives a trajectory $s_{x_i^*}$ with a smaller robustness than $s_{x_{i-1}^*}$: $f(x_i^*) < f(x_{i-1}^*)$. Thus $(f(x_i))_{i \in \mathbb{N}}$ is a decreasing sequence, lower bounded by 0. Therefore, it converges to a limit $r \geq 0$. But how to prove that this limit is indeed a minimum of f ?

Proposition 8. *Assume that Algorithm 1 halts at a point $z_Q = (x_Q, t_Q, \nu_Q)$, for which there exist \bar{t}_1, \bar{t}_2 such that:*

- $0 \leq \bar{t}_1 \leq t_Q \leq \bar{t}_2$
- $d_{\mathcal{U}}(s_{x_Q}(t)) > f(x_Q) \forall t \in T_R \triangleq [0, \bar{t}_1] \cup [\bar{t}_2, T)$, and
- $\bar{t}_2 - \bar{t}_1$ is ‘sufficiently small’.

Then x_Q is a local minimum of the robustness function f .

Proof. We assume that the trajectory starting at x_Q is safe - otherwise, we're done since we found an unsafe trajectory.

Two tubes will be constructed: one contains s_{x_Q} over (\bar{t}_1, \bar{t}_2) , the other contains it over T_R . They are such that no trajectory in them gets closer to \mathcal{W}_Q than s_{x_Q} . Then it is shown that all trajectories in a neighborhood of x_Q are contained in these tubes, making x_Q a local minimum of the robustness function f .

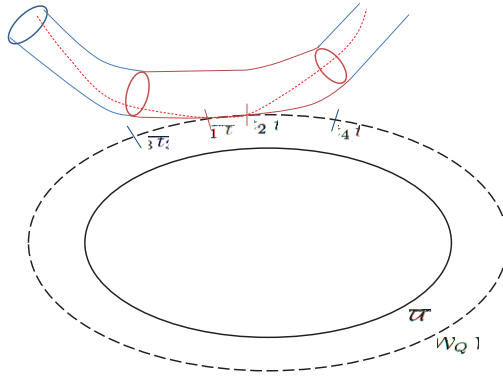


Fig. 4. [Proof of Prop.8] All trajectories starting in a neighborhood of x_Q will be contained in the orange tube over T_R and in the green tube over (\bar{t}_1, \bar{t}_2)

By the halting condition, $\nu_Q = 0$. Since the optimizer always returns a local minimum of the objective function F , there exists a neighborhood $N(z_Q)$ of z_Q such that for all $z \in N(z_Q)$, $F(z) \geq F(z_Q) = \nu_Q = 0$

$$\Leftrightarrow \forall (x, t, \nu) \in N(z_Q), s_x(t) \notin \text{int} \mathcal{W}_Q$$

$$\Leftrightarrow \forall (x, t, \nu) \in N(z_Q), d_{\mathcal{U}}(s_x(t)) \geq f(x_Q)$$

$N(z_Q)$ can be expressed as

$$N(z_Q) = B(x_Q, \epsilon) \times (\bar{t}_3, \bar{t}_4) \times (-\nu_l, \nu_l)$$

$$\epsilon > 0, \nu_l > 0, B(x_Q, \epsilon) \subset E(x_0) \cap X_0$$

(Since $\mathbb{R}^n \times \mathbb{R}_+ \times \mathbb{R}$ is a finite product, the box and product topologies are equivalent, so it doesn't matter which one we use.)

We now precise the notion of 'small enough': we require that

$$(\bar{t}_1, \bar{t}_2) \subseteq (\bar{t}_3, \bar{t}_4)$$

Therefore

$$\forall x \in B(x_Q, \epsilon), t \in (\bar{t}_1, \bar{t}_2), d_{\mathcal{U}}(s_x(t)) \geq f(x_Q)$$

Thus

$$\forall x \in B(x_Q, \epsilon), \inf_{t \in (\bar{t}_1, \bar{t}_2)} d_{\mathcal{U}}(s_x(t)) \geq f(x_Q) \quad (6)$$

We now study the behavior of trajectories starting in $B(x_Q, \epsilon)$ over the remaining time period T_R . Recall that $\mathcal{U} \subset \mathcal{W}_Q$. Let w^o be any point on the boundary $\partial\mathcal{W}_Q$. Then

$$\begin{aligned} \forall t \in T_R, d_{\mathcal{U}}(s_{x_Q}(t)) &> f(x_Q) = d_{\mathcal{U}}(w^o) > 0 \\ \Rightarrow \forall t \in T_R, d_{\mathcal{W}_Q}(s_{x_Q}(t)) &> 0 \end{aligned}$$

Then

$$\Lambda = \inf\{d_{\mathcal{W}_Q}(s_{x_Q}(t)) | t \in T_R\} > 0$$

s_x is continuous as a function of x for every t , therefore

$$\exists \delta > 0 \text{ s.t. } x \in B(x_Q, \delta) \Rightarrow d(s_{x_Q}(t), s_x(t)) < \Lambda$$

Pick any point $w \in \mathcal{W}_Q$. Then $\forall x \in B(x_Q, \delta)$ and $t \in T_R$

$$\begin{aligned} d(s_{x_Q}(t), w) &\leq d(s_{x_Q}(t), s_x(t)) + d(s_x(t), w) \\ &< \Lambda + d(s_x(t), w) \\ \Rightarrow d(s_x(t), w) &> d(s_{x_Q}(t), w) - \Lambda \end{aligned}$$

Minimizing both sides over $w \in \mathcal{W}_Q$,

$$\begin{aligned} d_{\mathcal{W}_Q}(s_x(t)) &> d_{\mathcal{W}_Q}(s_{x_Q}(t)) - \Lambda \geq 0 \\ \Rightarrow \inf_{t \in T_R} d_{\mathcal{W}_Q}(s_x(t)) &\geq 0 \end{aligned}$$

In conclusion

$$\forall x \in B(x_Q, \delta), \inf_{t \in T_R} d_{\mathcal{U}}(s_x(t)) \geq f(x_Q) \quad (7)$$

Putting Eqs.(6) and (7) together, it comes that $\forall x \in B(x_Q, \min\{\epsilon, \delta\})$

$$\begin{aligned} \inf_{t \in \mathbb{R}_+} d_{\mathcal{U}}(s_x(t)) &\geq f(x_Q) \\ \Leftrightarrow \forall x \in B(x_Q, \min\{\epsilon, \delta\}), f(x) &\geq f(x_Q) \end{aligned}$$

and x_Q is a local minimum of the robustness f .

Algorithm 2 RED with Simulated Annealing (SA+RED)

Input: An initial point $x \in X_0$.

Output: Samples $\Theta \subset X_0$.

Initialization: BestSoFar = x , $f_b = f(\text{BestSoFar})$

```
1: while  $f(x) > 0$  do
2:    $x' = \text{ProposalScheme}(x)$ 
3:    $\alpha = \exp(-\beta(f(x') - f_b))$ 
4:   if  $U(0, 1) \leq \alpha$  then
5:      $x^* = \text{RED}(x')$ 
6:      $x = x^*$ 
7:   else // Use the usual acceptance criterion
8:      $\alpha = \exp(-\beta(f(x') - f(x)))$ 
9:     if  $U(0, 1) \leq \alpha$  then  $x = x'$ 
10:    end if
11:  end if
12:   $(\text{BestSoFar}, f_b) = \text{BetterOf}(x, \text{BestSoFar})$ 
13: end while
```

4.3 Ellipsoid Descent with Stochastic Falsification

As outlined in the introduction, the proposed method can be used as a subroutine in a higher-level stochastic search falsification algorithm. A stochastic search will have a ProposalScheme routine: given a point x in the search space, ProposalScheme will propose a new point x' as a falsification candidate. Robustness Ellipsoid Descent (RED) may then be used to further descend from some judiciously chosen proposals. Algorithm 2 illustrates the use of RED within the Simulated Annealing (SA) stochastic falsification algorithm of [12]. $U(0, 1)$ denotes a number drawn uniformly at random over $(0, 1)$. Given two samples x and y , BetterOf(x, y) returns the sample with smaller robustness, and its robustness.

For each proposed sample x' , it is *attempted* with certainty if its robustness is less than the smallest robustness f_b found so far. Else, it is attempted with probability $e^{-\beta(f(x') - f_b)}$ (lines 3-4). If x' is attempted, RED is run with x' as starting point, and the found local minimum is used as final accepted sample (line 6). If the proposed sample is not attempted, then the usual acceptance-rejection criterion is used: accept x' with probability $\min\{1, e^{-\beta(f(x') - f(x))}\}$. As in the original SA method, ProposalScheme is implemented as a Hit-and-Run sampler (other choices can be made). The next section presents experimental results on three benchmarks.

4.4 Experiments

This section describes the experiments used to test the efficiency and effectiveness of the proposed algorithm SA+RED, and the methods compared against it.

We chose 3 navigation benchmarks from the literature: Nav0 (4-dimensional with 16 locations) is a slightly modified benchmark of [20], and it is unknown

Fig. 5. The navigation benchmark example.

whether it is falsifiable or not. Nav1 and Nav2 (4-dimensional with 3 locations) are the two hybrid systems in the HSolver library of benchmarks [21], and are falsifiable. We also chose a filtered oscillator, Fosc (32-dimensional with 4 locations), from the SpaceEx library of benchmarks [22]. We describe the Nav0 benchmark that we used, as it a slightly modified version of the benchmark in [20].

Example 3 (Navigation Benchmark [20]) *The benchmark studies a hybrid automaton \mathcal{H} with a variable number of discrete locations and 4 continuous variables x_1, x_2, y_1, y_2 that form the state vector $x = [x_1 \ x_2 \ y_1 \ y_2]^T$. The structure of the hybrid automaton can be better visualized in Fig. 5. The invariant set of every (i, j) location is an 1×1 box that constraints the position of the system, while the velocity can flow unconstrained. The guards in each location are the edges and the vertices that are common among the neighboring locations.*

Each location has affine constant dynamics with drift. In detail, in each location (i, j) of the hybrid automaton, the system evolves under the differential equation $\dot{x} = Ax - Bu(i, j)$ where the matrices A and B are

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1.2 & 0.1 \\ 0 & 0 & 0.1 & -1.2 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 & 0 \\ -1.2 & 0.1 \\ 0.1 & -1.2 \end{bmatrix}$$

and the input in each location is

$$u(i, j) = [\sin(\pi C(i, j)/4) \ \cos(\pi C(i, j)/4)]^T.$$

The array C is one of the two parameters of the hybrid automaton that the user can control and it defines the input vector in each discrete location. Here, we consider the input array denoted in Fig. 5.

The set of initial conditions is the set $H_0 = \{13\} \times [0.2 \ 0.8] \times [3.2 \ 3.8] \times [-0.4 \ 0.4]^2$ (green box in Fig. 5) and the unsafe set is $\mathcal{U} = \{4\} \times \{x \in \mathbb{R}^4 \mid \|x - (3.5 \ 0.5 \ 0 \ 0)\| \leq 0.3\}$ (red circle in Fig. 5). This is slightly modified from the original benchmark to simplify the programming of the p_i functions. Sample trajectories of the system appear in 5 for initial conditions $[0.8 \ 3.2 \ -0.2 \ 0.35]^T$ (red trajectory) and $[0.4 \ 3.3 \ -0.1 \ -0.1]^T$ (blue trajectory). Note that the two trajectories follow different discrete locations.

The methods compared are: SA+RED, pure Simulated Annealing (SA) [12], mixed mode-HSolver (mm-HSolver) [21], and the reachability analysis tool SpaceEx [22]. Because ours is a falsification framework, SpaceEx is used as follows: for a given bound j on the number of discrete jumps, SpaceEx computes an *over-approximation* $\overline{R}(j)$ of the set reachable in j jumps $R(j) : R(j) \subset \overline{R}(j)$. If $\overline{R}(j) \cap \mathcal{U}$ is empty, then *a fortiori* $R(j) \cap \mathcal{U}$ is empty, and the system is safe if trajectories are restricted to j jumps. If, however, $\overline{R}(j) \cap \mathcal{U} \neq \emptyset$, no conclusion can be drawn.

Because SA and SA+RED are stochastic methods, their behavior will be studied by analyzing a number of runs. A *regression* will mean a fixed number of runs, all executed with the same set of parameters, on the same benchmark. mm-HSolver is deterministic, and thus one result is presented for benchmarks Nav1 and Nav2 (Nav0 was not tested by mm-HSolver’s authors [21]). The mm-HSolver results are those reported in the literature. SpaceEx was run in deterministic mode on Nav0 (specifically, we set parameter ‘directions’ = ‘box’ [22]).

Parameter setting: We set the test duration $T = 12\text{sec}$, which we estimate is long enough to produce a falsifying trajectory for Nav0 if one exists. For SA+RED, we chose to generate 10 samples ($|\Theta| = 10$). We will see that even this small number is enough for the algorithm to be competitive. A regression consists of 20 jobs. The SpaceEx parameters were varied in such a way that the approximation \overline{R} of the reachable set R became increasingly precise. Clustering% was given the values 0, 20 and 80 (the smaller the Clustering%, the better the approximation and the longer the runtime). The ODE solver timestep δ was given the values 0.0008, 0.02, 0.041 seconds. These are, respectively, the minimum, median, and average values of δ used by the variable step-size ODE solver used by SA+RED. The smaller δ , the better the approximation and the longer the runtime. The following parameters were fixed: ‘directions’ = ‘box’, ‘Local time horizon’ = 10sec, rel-err = abs-err = 1.0e-10. The Nav0 SpaceEx configuration files can be obtained by request from the authors.

The performance metrics: Each run produces a minimum robustness. For a given regression, we measure: the smallest, the average, and the largest minimum robustness found by the regression (min, avg, max in Table 1). The standard deviation of minimum robustness is also reported (σ_f). For SpaceEx, we had to simply assess whether $\overline{R}(j)$ intersected \mathcal{U} or not.

The cost metric: Each run also counts the number of simulated trajectories in the course of its operation: SA simulates a trajectory for each proposed sample, SA+RED simulates a trajectory each time the constraint function of Prob3[W_i] is evaluated (and for each sample), and mm-HSolver simulates tra-

jectories in falsification mode. The trajectories simulated by SA and SA+RED have a common, fixed, pre-determined duration T . Thus the cost of these algorithms can be compared by looking at the Number of Trajectories (NT) each simulates (column \overline{NT} in Table 1 - the overline denotes an average). The trajectories computed by mm-HSolver have varying lengths, determined by a quality estimate. So for comparison, we report the number of single simulation steps (SS), i.e. the number of points on a given trajectory (column \overline{SS} - mm-HSolver, being deterministic, has one value of SS). Unfortunately, SS doesn't include the cost of doing verification in mm-HSolver, so it should be considered as a *lower bound* on its computational cost. On the other hand, because of the choice of T , the SS numbers reported for SA+RED should be treated as *upper bounds*: choosing a shorter a-priori T will naturally lead to smaller numbers. An exact comparison of the costs of SA+RED and mm-HSolver would require knowing the duration of the shortest falsifying trajectory, and setting the a-priori T to that, and somewhat incorporating the cost of verification. The operations that SpaceEx does are radically different from those of the other methods compared here. The only way to compare performance is through the runtime.

Experimental setup: we impose an upper limit NT_{MAX} on NT : SA+RED is aborted when its NT reaches this maximum, and SA is made to generate NT_{MAX} samples. (Of course, SA+RED might converge before simulating all NT_{MAX} trajectories). 3 values were chosen for NT_{MAX} : 1000, 3000 and 5000. For each value, a regression is run and the results reported. This allows us to measure the competitiveness of the 2 algorithms (i.e. performance for cost).

Experiments: Table 1 compares SA+RED to SA: we start by noting that SA+RED falsified Nav2, whereas SA failed to so. On most regressions, SA+RED achieves better performance metrics than SA, for the same (or lower) computational cost. This is consistent whether considering best case (min), average case (avg) or worst case (max). There are 2 exceptions: for Nav1 and Nav2, $NT_{MAX} = 5000$ produces better average and max results for SA than for SA+RED. When running realistic system models, trajectory simulation is the biggest time consumer, so effectively NT is the limiting factor. So we argue that these 2 exceptions don't invalidate the superiority of SA+RED as they occur for high values of NT that might not be practical with real-world models. (In these cases, we observed that SA eventually produces a sequence of samples whose trajectories finish by making a large number of jumps between locations 3 and 2, with a relatively high robustness. From there SA then produces a sample with 0 (or close to 0) robustness. This happens on every Nav1 run we tried, and most Nav2 runs, resulting in the numbers reported. The RED step in SA+RED seems to avoid these trajectories by 'escaping' into local minima, and is worthy of further study.)

Table 2 compares SA+RED to mm-HSolver. We note that SA+RED falsifies the benchmarks, as does mm-HSolver. For Nav1, \overline{SS} is greater than mm-HSolver's SS , though the falsifying runs have SS values (last column) both smaller and larger than mm-HSolver. For Nav2, which appears to be more chal-

System	NT_{MAX}	NT (σ_{NT})	σ_f	SA+RED Rob. min, avg, max	SA Rob. min, avg, max
Nav0	1000	1004 (1.4)	0.022	0.2852, 0.30,0.35	0.2853,0.33,0.33
	3000	2716 (651)	0.019	0.2852,0.29,0.32	0.2858,0.31,0.36
	5000	4220 (802)	0.009	0.285,0.28,0.32	0.286,0.32,0.35
Nav1	1000	662 (399)	0.21	0,0.43,0.65	0,0.96,1.88
	3000	1129 (1033)	0.23	0,0.39,0.65	0,0.99,1.80
	5000	1723 (1770)	0.23	0,0.38,0.68	0,0,0
Nav2	1000	902 (246)	0.32	0,0.54,0.78	0.3089,1.11,1.90
	3000	1720 (1032)	0.3	0,0.53,0.83	0.3305,1.29,1.95
	5000	1726 (1482)	0.27	0,0.62,0.79	0,0.002,0.01
Fosc	1000	1000 (9.3)	0.024	0.162,0.206,0.251	0.1666,0.216,0.271
	3000	3000 (8.7)	0.024	0.163,0.203,0.270	0.173,0.212,0.254
	5000	5000 (11)	0.028	0.167,0.193,0.258	0.185, 0.218, 0.245

Table 1. Comparison of SA and SA+RED. To avoid clutter, Robustness values are reported to the first differing decimal, with a minimum of 2 decimals. σ_f is standard deviation of robustness for SA+RED.

System	NT_{MAX}	SS (σ_{SS})	σ_f	SA+RED Rob min, avg, max	mm-HSolver Rob, NT , SS	SS at min Rob for SA+RED
Nav1	1000	47k (30k)	0.21	0,0.43,0.65	0, 22,5454	0,1560,16k
	3000	79k (76k)	0.23	0,0.39,0.65		0,0,1600, 127k
	5000	143k (141k)	0.23	0,0.38,0.68		7660, 38k, 102k, 159k
Nav2	1000	63k (18k)	0.32	0,0.54,0.78	0, 506, 138k	2888, 74k
	3000	126k (80k)	0.3	0,0.53,0.83		14k, 57k, 210k
	5000	124k (114k)	0.27	0,0.622,0.79		3450, 121k, 331k

Table 2. Comparison of SA+RED and mm-HSolver. The last column shows some of the SS values at which min robustness is achieved by SA+RED on various runs.

lenging, SA+RED performed better on average than mm-HSolver. However, we point out again that exact comparison is hard.

For SpaceEx running on Nav0, we observed that our initial parameter set produces an $\overline{R(j)}$ that intersects \mathcal{U} . Since this is inconclusive, we modified the parameters to get a better approximation. For parameter values (Clustering%, δ) = (0, 0.0008), $\overline{R(j)}$ and \mathcal{U} were almost tangent, but SpaceEx runtimes far exceeded those of SA+RED (more than 1.5 hours). Moreover, SpaceEx did not reach a fixed point of its iterations (we tried up to $j = 200$ iterations before stopping due to high runtimes). Thus, we can not be sure that all of the reachable space was covered. While this may be seen as an analogous problem to the choice of T in SA+RED, the computational cost of increasing j is much more prohibitive than that of increasing T . We now present some detailed runtime results. For SA+RED, ‘runtime’ means the User time reported by the Unix *time* utility. SA+RED was run on a dedicated Intel Xeon processor, x86-64 architecture, under the Unix OS. SpaceEx reports its own runtime. It was run on a Dual-

Clustering%	$\delta(\text{sec})$			SA+RED Runtime (sec) min,avg,max	NT_{MAX}
	0.0008	0.002	0.041		
80	737	30	15	324, 426, 596	1000
20	1066	53	33	620, 1132, 1385	3000
10	1460	NA	NA	767,1617, 2216	5000
0	> 5400	NA	NA		

Table 3. Comparison of SA+RED and SpaceEx runtimes. NA means the experiment was not run, because a more accurate run was required. The right-most columns shows the NT_{MAX} constraint for which the SA+RED runtimes were obtained.

Core Intel Centrino processor, under a Windows7 64b OS, with no other user applications running.

Thus we may conclude that stochastic falsification and reachability analysis can play complementary roles in good design practice: first, stochastic falsification computes the robustness of the system with respect to some unsafe set. Guided by this, the designer may make the system more robust, which effectively increases the distance between the (unknown) reachable set and the unsafe set. Then the designer can run a reachability analysis algorithm where coarse over-approximations can yield conclusive results.

5 Conclusions

The minimum robustness of a hybrid system is an important indicator of how safe it is. In this paper, we presented an algorithm for computing a local minimum of the robustness for a certain class of linear hybrid systems. The algorithm can also be used to minimize the robustness of non-hybrid linear dynamic systems. When integrated with a higher-level stochastic search algorithm, the proposed algorithm has been shown to perform better than existing methods on literature benchmarks, and to complement reachability analysis. We will next deploy this capability to perform local descent search for the falsification of arbitrary linear temporal logic specifications, not only safety specifications. This investigation opens the way to several interesting research questions. Most practically, reducing the number of tests NT results in an immediate reduction of the computation cost. Also useful, is the determination of an appropriate test duration T , rather than a fixed arbitrary value.

In terms of performance guarantees, obtaining a lower bound on the optimum achieved in Problem 3 could lead to a lower bound on the optimal robustness. One level higher in the algorithm, it is important to get a theoretical understanding of the behavior of the Markov chains iterated by SA+RED to further improve it.

References

1. Girard, A., LeGuernic, C.: Efficient reachability analysis for linear systems using support functions. In: IFAC World Congress. (2008) 22–35

2. Asarin, E., Dang, T., Maler, O., Testylier, R.: Using redundant constraints for refinement. In: International Symposium on Automated Technology for Verification and Analysis. Volume 6252 of LNCS., Springer (2010)
3. LeGuernic, C., Girard, A.: Reachability analysis of hybrid systems using support functions. In: Computer Aided Verification. Volume 5643 of LNCS., Springer (2009) 540–554
4. Althoff, M., Stursberg, O., Buss, M.: Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear Analysis: Hybrid Systems* **4**(2) (2010) 233 – 249
5. Girard, A., Pappas, G.J.: Verification using simulation. In: Hybrid Systems: Computation and Control (HSCC). Volume 3927 of LNCS., Springer (2006) 272 – 286
6. Julius, A.A., Fainekos, G., Anand, M., Lee, I., Pappas, G.: Robust test generation and coverage for hybrid systems. In: Hybrid Systems: Computation and Control. Volume 4416 of LNCS., Springer-Verlag Berlin Heidelberg (2007) 329–342
7. Dang, T., Donze, A., Maler, O., Shalev, N.: Sensitive state-space exploration. In: Proc. of the 47th IEEE Conference on Decision and Control. (2008) 4049–4054
8. Branicky, M., Curtiss, M., Levine, J., Morgan, S.: Sampling-based planning, control and verification of hybrid systems. *IEE Proc.-Control Theory Appl.* **153**(5) (2006) 575–590
9. Plaku, E., Kavradi, L.E., Vardi, M.Y.: Falsification of ltl safety properties in hybrid systems. In: Proc. of the Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Volume 5505 of LNCS. (2009) 368 – 382
10. Rizk, A., Batt, G., Fages, F., Soliman, S.: On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In: International Conference on Computational Methods in Systems Biology. Number 5307 in LNCS, Springer (2008) 251–268
11. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to simulink/stateflow verification. In: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control. (2010) 243–252
12. Nghiem, T., Sankaranarayanan, S., Fainekos, G., Ivancic, F., Gupta, A., Pappas, G.: Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In: Hybrid Systems: Computation and Control. (2010)
13. Fainekos, G., Pappas, G.: Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* **410**(42) (2009) 4262–4291
14. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings of the 11th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press (1996) 278–292
15. Boyd, S., Vandenberghe, L.: *Convex Optimization*. Cambridge University Press (2004)
16. Han, Z.: Formal Verification of Hybrid Systems using Model Order Reduction and Decomposition. PhD thesis, Dept. of ECE, Carnegie Mellon University (2005)
17. Tabuada, P.: *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer (2009)
18. Julius, A.A., Pappas, G.J.: Trajectory based verification using local finite-time invariance. In: Hybrid Systems: Computation and Control. Volume 5469 of LNCS., Springer (2009) 223–236
19. Lygeros, J., Johansson, K.H., Simic, S.N., Zhang, J., Sastry, S.: Dynamical properties of hybrid automata. *IEEE Transactions on Automatic Control* **48** (2003) 2–17

20. Fehnker, A., Ivancic, F.: Benchmarks for hybrid systems verification. In: Hybrid Systems: Computation and Control. Volume 2993 of LNCS., Springer (2004) 326–341
21. Ratschan, S., Smaus, J.G.: Finding errors of hybrid systems by optimizing an abstraction-based quality estimate. In: Proceedings of the Third Int'l Conf. on Tests and Proofs, Zurich, Switzerland (2009) 153–168
22. Frehse, G., Guernic, C.L., Donz, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Proceedings of the 23d CAV. (2011)